Université de Versailles
IUT de Vélizy - Algorithmique

Vanessa Vitse
2007/2008

TP n°6

(2 séances)

Pour vous éviter d'avoir à recopier les programmes, tous les listings se trouvent dans l'archive `listingsTP6.zip` sur e-campus.
Tous les exercices doivent être traités, les réponses attendues doivent contenir des explications concises de vos résultats. La clarté de votre rédaction, ainsi que l'orthographe seront pris en compte dans la note.
Vous rédigerez votre compte rendu dans un fichier type Word (extension `.doc` ou `.odt`) que vous déposerez sur e-campus avant la date limite dans l'espace prévu à cet effet. Aucun retard ne sera excusé.

Avant de commencer ce TP, **vous devez avoir terminé le TP5 et posté votre compte rendu sur e-campus**.

Rappels sur les méthodes

1. Les définitions des méthodes doivent être écrites dans le corps de la classe, ie après la ligne `class NomClasse{` et avant le } en fin de fichier.
2. Comme les noms de variables, les noms des méthodes commencent par une minuscule, contrairement aux noms des classes qui commencent toujours par une majuscule.
3. La syntaxe d'une méthode est la suivante :

   ```
   static <type retour> nomMethode (<type param1> <nomParam1>,...){
   <instructions>
   ...
   }
   ```

   Par exemple, pour définir la méthode `carre` qui calcule le carré d'un nombre entier, on écrira :

   ```
   static int carre(int x){
   return x*x;
   }
   ```

4. Une méthode peut donc
   – prendre plusieurs paramètres (de même type ou de type différents)
     ex : `String substring(int beginIndex, int endIndex)`, `int lastIndexOf(String str, int fromIndex)`
   – ne pas prendre de paramètre *ex :* `int intValue()`
   – renvoyer une valeur (cf les exemples ci-dessus)
   – ne pas renvoyer de valeur *ex :* `void main(String[] args)`

5. Une méthode qui est définie dans le fichier courant, ou qui a été importée en début de fichier en même temps qu'une classe ou un package (*ex :* `import java.util.*`), peut être utilisée à tout moment (on dit qu'on *appelle* la méthode). La ligne static <type retour> nomMethode (<type param1> <nomParam1>,...) désigne la *signature* de la méthode : il est essentiel de la connaître lorsqu'on veut appeler la méthode puisqu'elle donne le nombre et le type des paramètres à passer en argument, ainsi que le type de la valeur de retour (la signature est en quelque sorte le mode d'emploi de la méthode).
   ex : La méthode `carre` a la signature suivante : `static int carre(int x)`
   Donc pour calculer le carré de 5 et l'assigner à la variable `c`, on écrit :

```
c = carre(5);
```

Exercice 1.

1. Ecrire une méthode `max2` qui calcule le maximum de deux nombres entiers.
2. Tester cette méthode avec le programme suivant :

```
import java.util.*;

class Max2{
    /*votre definition pour max2*/

    public static void main(String[] argv){
        Scanner sc = new Scanner(System.in);
        System.out.println("Entrer deux nombres entiers : ");
        System.out.print("x = ");
        int x = sc.nextInt();
        System.out.print("y = ");
        int y = sc.nextInt();
        System.out.println("Le maximum de x et y vaut " + max2(x,y));
    }
}
```

3. Ecrire une méthode `max3` qui calcule le maximum de trois nombres entiers sans utiliser de conditionnelle (`if ... else ...`). Ecrire un programme appelant cette méthode.

Exercice 2. Qu'affichent les deux programmes suivants ? Expliquer.

```
class Echange{
    public static void main(String[] args){
        int i = 10;
        int j = 1;
        System.out.println("Avant echange :\n i = " + i + "; j = " + j);
        int k;
        k=i; i=j; j=k;
        System.out.println("Apres echange :\n i = " + i + "; j = " + j);
    }
}

class MauvaisEchange{
    public static void main mauvaisEchange(int a, int b){
        int c;
        c = a;
```

```java
        a = b;
        b = c;
    }
    public static void main(String[] args){
        int i = 10;
        int j = 1;
        System.out.println("Avant echange : \n i = " + i + "; j = " + j);
        mauvaisEchange(i,j);
        System.out.println("Apres echange : \n i = " + i + "; j = " + j);
    }
}
```

Exercice 3. Utilisation de boucle `for`

1. Ecrire une méthode `static void ligne(int n)` qui affiche une ligne contenant n symboles *.
2. Ecrire une méthode `static carrePlein(int n)` qui affiche un carré plein de n par n.
3. Ecrire une méthode `static void carreVide(int n)` qui affiche un carre vide de n par n.
4. Ecrire une méthode `static int triangle(int n)` qui affiche un triangle plein de hauteur n et de base n.

```
carrePlein(10)  carreVide(10)   triangle(10)
**********      **********      *
**********      *        *      **
**********      *        *      ***
**********      *        *      ****
**********      *        *      *****
**********      *        *      ******
**********      *        *      *******
**********      *        *      ********
**********      *        *      *********
**********      **********      **********
```

Exercice 4.

1. Ecrire une méthode `static double puissance(double nb, int n)` qui calcule la puissance $n^{\text{ème}}$ du nombre `nb`, où `n` est un entier positif.
2. Modifier la méthode précédente pour traiter le cas où `n` est un entier quelconque.
3. Tester la méthode `puissance` en calculant par exemple les puissances suivantes :

$$2.5^3 \qquad 2.5^0 \qquad 2.5^{-3}$$

Exercice 5. Ecrire une méthode `static int factorielle(int n)` qui calcul la factorielle d'un entier `n` positif ou nul.

Tester votre méthode en écrivant une méthode `main`.
Pour mémoire, $n!$ est défini récursivement de la façon suivante :

$$\begin{cases} 0! = 1 \\ n! = n \times (n-1)! \end{cases}$$

Exercice 6. Ecrire une méthode `static int somCarresPairs(int n)` qui calcule la somme des carrés des `n` premiers entiers pairs.
ex : `somCarresPairs(5)` $= 0^2 + 2^2 + 4^2 + 6^2 + 8^2 = 120$

Exercice 7. On définit la suite double (qui n'est pas la suite arithmético-géométrique)

$$\begin{cases} u_0 = 1 \ ; \ v_0 = 2 \\ u_{n+1} = \frac{u_n + v_n}{2} \ ; \ v_{n+1} = \sqrt{u_{n+1} v_n} \end{cases}$$

On admet que les suites (u_n) et (v_n) sont adjacentes de limite $\dfrac{\sqrt{27}}{\pi}$.

1. Ecrire une méthode `static double approximationDePi(int n)` qui calcule l'approximation du nombre π obtenue à partir de v_n. On utilisera la méthode `Math.sqrt` inclue dans le package `java.lang`, qui permet de calculer la racine carrée d'un nombre.
2. Ecrire la méthode main permettant de tester la méthode précédemment écrite.

Exercice 8 Ecrire une méthode `static void triangle2(int n)` qui affiche un triangle comme celui dessiné ci-dessous lorsque `n = 5`.



```
    *
   ***
  *****
 *******
*********
```

Exercice 9. La suite Syracuse(p) est définie par récurrence de la façon suivante :

$$\begin{cases} a_0 = p \\ a_{n+1} = \begin{cases} \frac{a_n}{2} & \text{si } a_n \text{ est pair} \\ 3a_n + 1 & \text{si } a_n \text{ est impair} \end{cases} \end{cases}$$

1. Ecrire une méthode `static int syracuse(int p, int n)` qui calcule la valeur du $n^{\text{ème}}$ terme de la suite Syracuse(p).
2. Ecrire une méthode `static int indice(int p)` qui calcule le plus petit n tel que le terme a_n de la suite Syracuse(p) soit égal à 1.